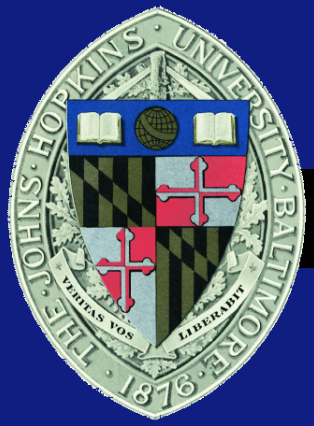
Lecture 5.3
Open MP Loops

EN 600.320/420

Instructor: Randal Burns

13 February 2017



Department of Computer Science, *Johns Hopkins University*

A Loop OpenMP Program

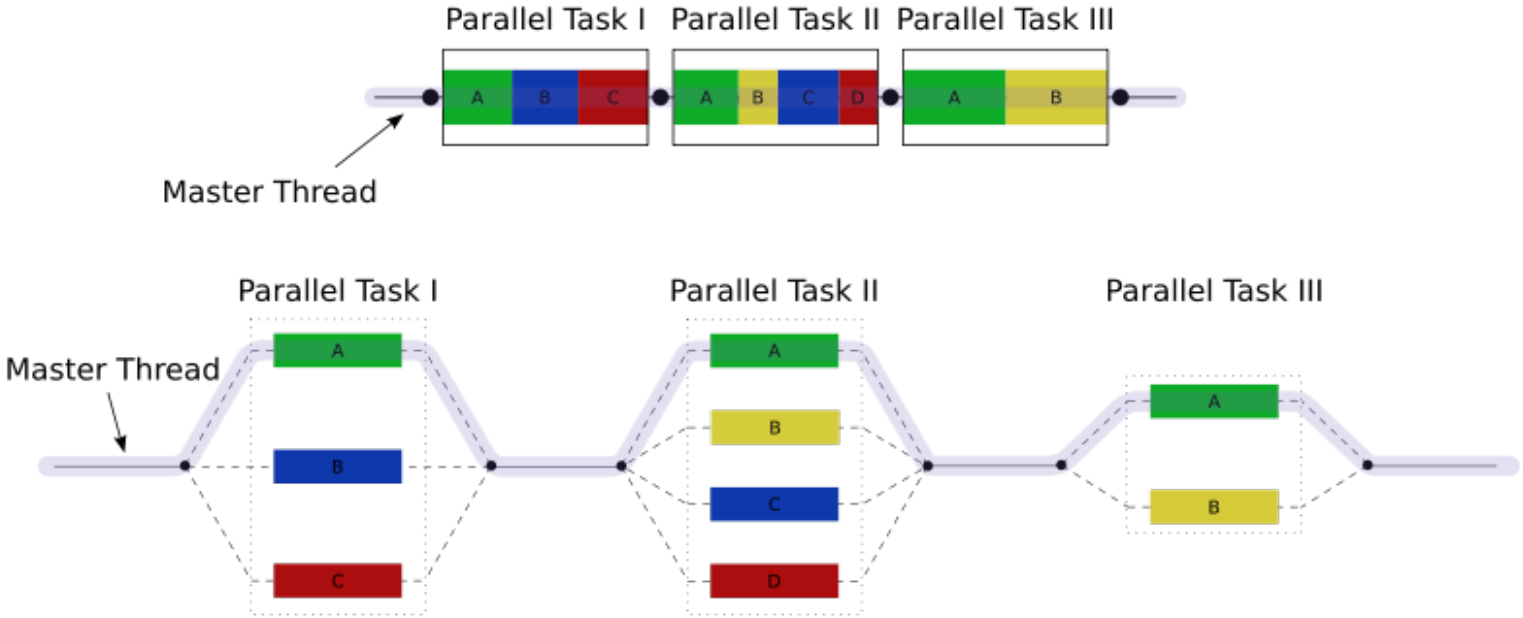
- See `loop.c`

- Parallelization directive
  - `#pragma omp parallel for`
  - that executes a block of C/C++/Fortan code in parallel

- OpenMP maps loop iterations to threads
  - Same threads may run many loop iterations
  - Serial equivalence ensures all configurations are the same

OpenMP is Master/Worker

- Yeah strictly
  - In that there's an entry thread that runs in the serial context and creates other threads
- The wiki page has this figure



http://en.wikipedia.org/wiki/File:Fork_join.svg

OpenMP is Loop Parallelism

- When used with loops (most common)
  - I think of it in this regard
- Why am I making a big deal of this?
  - Because the book (Ch. 5) says that OpenMP doesn't work that well for Master/Worker and they have a point.
  - Limited messaging support.
  - Synchronization constructs are confusing.

Loop Scheduling

#pragma omp parallel for schedule(kind [,chunk size])

- Chunk size = number of iterations per thread
- Directives that tell OpenMP how to schedule chunks
  - Static – divide loop into equal sized chunks
  - Dynamic—build internal work queue and dispatch blocksize at a time
  - Guided—dynamic scheduling with decreasing block size for load balance
  - Auto—compiler chooses from above
  - Runtime—runtime configuration chooses from above

Loop Parallelism

Most (science and engineering) codes follow this pattern

- Most frequently implemented pattern for shared-memory

Reasons people choose loop parallelism

- **Sequential equivalence:** parallel program is equivalent to a serial program (easy to write and maintain, good tools)
- **Refactoring**: Incremental conversion of a serial program to a parallel program (easy to test and debug)

Forces against

- **Memory utilization:** if loop access patterns don't match cache hierarchy, program often require massive restructuring (works against seq. equiv. and refactoring)